Cat-Dog Classification

**Objective**

This would be the binary classification model as we just have to classify image between two featured outputs only CAT or DOG.

**Procedure**

The first step is to import required library and load dataset. On these datasets, the shape does not content image size and number of channels, so we must do resizing according to our need.

As values in array are in range from 0 to 255 but to train our model appropriately, we need to rescale these values between 0 to 1 so we divide training and test dataset by 255.

We start to create our model which is Sequential model because in this model the layers are going to be stacked up in the sequence. Sequential model consists of array of different layers.

First the Convolutional layer and its Pooling,

Conv2D(32, (3, 3), activation = 'relu', input\_shape = (100, 100, 3))

Conv2D is a function with first parameter is the number of filters we want to use and second parameter is the size of that filter.

Then we have to define which activation function we going to use. For first layer we must have to mention input shape which means what kind of shape is this model expecting, for this model we expecting 100x100x3 size image.

MaxPooling2D((2, 2))

It just contains the filter size.

Similarly, we can add multiple convolutional layer but from second layer we does not required to put input shape.

Conv2D(32, (3, 3), activation = 'relu')

MaxPooling2D((2,2))

Now we Flatten our model means converting into 1D and making fully connected layer using Dense()

Dense(64, activation = 'relu')

Here, first parameter is the number of neurons we want to keep in first fully connected layer and second parameter is the activation function we are using.

Dense(1, activation = 'sigmoid')

Final fully connected which is our output layer.

Output layer must have the same number of neurons as output class but for binary classification we only need one output neuron.

Activation function is sigmoid because it is binary classification in the output layer.

We now have to compile our model by adding the loss and the back propagation.

model.compile(loss = 'binary\_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

We use “binary\_crossentropy” loss because we are implementing binary classification and metrics indicate a metric on which we want to evaluate our model

Training our model,

model.fit(X\_train, y\_train, epochs = 5, batch\_size = 64)

Train by giving training dataset we have already passing epochs and batch size.

Evaluating our model,

model.evaluate(X\_test, y\_test)

**Testing**

First, we will find some random integer between range to the size of test dataset and display image on this index.

index = random.randint(0, len(X\_test))

plt.imshow(X\_test[index, :])

plt.show()

y\_pred = model.predict(X\_test[index, :].reshape(1, 100, 100, 3))

print('probability of image: ', y\_pred)

y\_pred = y\_pred > 0.5

if(y\_pred == 0):

    pred = 'dog'

else:

    pred = 'cat'

print('According to model it is: ', pred)

Now we find the probability of that image using predict() function passing image on that index after resizing.

Making them classify as DOG or CAT on the basis of probability we obtain.